MACHINE LEARNING METHODS REVIEW FOR SENTIMENT ANALYSIS

2022

COMPUTER ENGINEERING

SENIOR PROJECT

Enes KURBETOĞLU

DUYGU ANALİZİ İÇİN MAKİNE ÖĞRENMESİ YÖNTEMLERİ İNCELEMESİ

Enes KURBETOĞLU

Karabük Üniversitesi

Mühendislik Fakültesi

Bilgisayar Mühendisliği Bölümünde

Bitirme Projesi Tezi

Olarak Hazırlanmıştır.

KARABÜK

NİSAN 2022

Enes KURBETOĞLU tarafından hazırlanan “DUYGU ANALİZİ İÇİN MAKİNE ÖĞRENMESİ YÖNTEMLERİ İNCELEMESİ” başlıklı bu projenin Bitirme Projesi Tezi olarak uygun olduğunu onaylarım.

Yrd. Doç. Dr. Yasin ORTAKÇI .....

Bitirme Projesi Danışmanı, Bilgisayar Mühendisliği Anabilim Dalı

...../ ...../2022

Bilgisayar Mühendisliği bölümü, bu tez ile, Bitirme Projesi Tezini onamıştır.

Dr. Öğretim Üyesi Yüksel ÇELİK .....

Bölüm Başkanı

“Bu projedeki tüm bilgilerin akademik kurallara ve etik ilkelere uygun olarak elde edildiğini ve sunulduğunu; ayrıca bu kuralların ve ilkelerin gerektirdiği şekilde, bu çalışmadan kaynaklanmayan bütün atıfları yaptığımı beyan ederim.”

Enes KURBETOĞLU

ABSTRACT

Senior Project Thesis

MACHINE LEARNING METHODS REVIEW FOR SENTIMENT ANALYSIS

Enes KURBETOĞLU

Karabuk University

Computer Engineering

Department of Computer Engineering

Project Supervisor:

Assist. Prof. Dr. Yasin ORTAKÇI

June 2022, 48 pages

Classification in machine learning and statistics is a supervised learning approach in which the computer program learns from the data given to it and make new observations or classifications. In this study this powerful method along with many other tools will be used to classify tweets posted since May 2020 until February 2022 and observe the change in public opinion about different COVID-19 vaccinations and vaccinations in general and compare various classification algorithms as well as comparing traditional machine learning methods with deep learning methods.

Keywords: Veri Madenciliği, Sınıflandırma, Makine öğrenmesi, Derin öğrenme, Yapar sinir ağları Text classification, Data mining, Machine learning, Deep learning, Neural networks.

TEŞEKKÜR

Bu tez çalışmasının planlanmasında, araştırılmasında, yürütülmesinde, oluşumunda ve gelişiminde ilgi ve desteğini esirgemeyen, engin bilgi ve tecrübelerinden yararlandığım, yönlendirme ve bilgilendirmeleriyle çalışmamı bilimsel temeller ışığında şekillendiren sayın hocam Yrd. Doç. Dr. Yasin ORTAKÇI'ya sonsuz teşekkürlerimi sunarım.

Table of Contents

ABSTRACT iv
TEŞEKKÜR v
INDEX OF FIGURES AND TABLES viii
CHAPTER ONE 1
INTRODUCTION 1
1.1 Intro and Goal 1
1.2 Literature Review 1
CHAPTER TWO 4
DATA 4
2.1 Overall Framework for Training 4
2.2 Dataset 5
2.3 Feature Selection 5
2.4 Data Cleaning 6
2.5 Visualisation of Data 6
2.6 Vectorisation 7
2.7 Sampling 10
CHAPTER THREE 12
MODEL 12
3.1 Naïve Bayes 12
3.1.1 Multinomial Naïve Bayes 12
3.1.2 Complement Naïve Bayes 12
3.2 Support Vector Machines 13
3.3 Logistic Regression 14
3.4 Decision Trees 16
3.4.1 Random Forest Classifier 16
3.4.2 Extreme Gradient Boosting (XGBoost) Classifier 17
3.5 Voting Classifier 17
3.6 Recurrent Neural Networks 18
3.6.1 Embedding Layer 19
3.6.2 Long Short-Term Memory Layer 20
3.6.3 Dense Layer 23
3.6.4 Gradient Descent 23
3.6.5 Backpropagation 24
3.6.6 RNN Architecture 24

CHAPTER FOUR 26
HYPER-PARAMETER TUNING AND CROSS-VALIDATION 26
4.1 Cross-Validation 26
4.2 Randomised Search 27
4.3 Grid Search 27
CHAPTER FIVE 28
MODEL EVALUATION 28
5.1 Metrics 28
5.1.1 Confusion Matrix 28
5.1.2 Accuracy 28
5.1.3 Precision 29
5.1.4 Recall 29
5.1.5 Specificity 29
5.1.6 F1 Score 29
5.1.7 ROC and AUC 29
CHAPTER SIX 31
EXPERIMENTAL STUDY 31
6.1. Training Results 31
6.1.1 Unigram Model 31
6.1.2 Bi-gram Model 32
6.1.3 ROC-AUC 33
6.1.4 RNN Model 34
6.2 Analysis on New Data 35
6.3 Discussion 44
CHAPTER SEVEN 46
CONCLUSION 46
REFERENCES 47

INDEX OF FIGURES AND TABLES

FIGURE 1 Overall framework for training 4
TABLE 1 Overview of Features 5
FIGURE 2 Class count in training dataset 6
FIGURE 3 Positive words cloud with their frequencies emphasised by size 7
FIGURE 4 Negative words cloud with their frequencies emphasised by size 7
TABLE 2 TF-IDF calculation example 10
FIGURE 5 SVM threshold representation 14
FIGURE 6 The sigmoid function 15
FIGURE 7 Representation of a decision tree 16
FIGURE 8 Voting classifier 18
FIGURE 9 Embedding layer visual 19
FIGURE 10 A simple RNN 20
FIGURE 11 An LSTM structure 20
FIGURE 12 The forget gate layer 21
FIGURE 13 The layers determining the candidate values to be stored 21
FIGURE 14 Update the cell state 22
FIGURE 15 The processed output of the cell 22
FIGURE 16 Gradient descent 24
FIGURE 17 RNN Architecture representation 25
FIGURE 18 Cross-validation 26
FIGURE 19 The confusion matrix 28
TABLE 3 Results for the unigram model 31
TABLE 4 Results for the bi-gram model 32
FIGURE 20 ROC-AUC values for the UNIGRAM model 33
FIGURE 21 ROC-AUC values for the BI-GRAM model 33
FIGURE 22 Accuracy and loss values for the RNN model 34
FIGURE 23 Results for the RNN model 34
FIGURE 24 Number of tweets analysed by month 35
FIGURE 25 Number of tweets analysed by vaccine 35
FIGURE 26 Results with MNB by month 36
FIGURE 27 Results with MNB by vaccine 36
FIGURE 28 Results with CNB by month 37
FIGURE 29 Results with CNB by vaccine 37
FIGURE 30 Results with LSVC by month 38

FIGURE 31 Results with LSVC by vaccine 38
FIGURE 32 Results with LRG by month 39
FIGURE 33 Results with LRG by vaccine 39
FIGURE 34 Results with RFC by month 40
FIGURE 35 Results with RFC by vaccine 40
FIGURE 36 Results with XGB by month 41
FIGURE 37 Results with XGB by vaccine 41
FIGURE 38 Results with VC-H by month 42
FIGURE 39 Results with VC-H by vaccine 42
FIGURE 40 Results with RNN by month 40
FIGURE 41 Results with RNN by vaccine 41

CHAPTER ONE

INTRODUCTION

1.1 Intro and Goal

The origin of COVID-19 is said to be in the starting of December 2019, when several patients from Wuhan, China reported severe respiratory infections. By the end of January, the virus had already started spreading out to other countries. Thus, many countries initiated lockdown orders. A lot of people expressed their opinion on the lockdown, the virus, and the vaccinations and its effects. There became more and more people opposed to vaccination; most of them feared vaccine's side effects, there were also some that conspired that this was a way of controlling the public. A lot of these concerns were brought up on Twitter. Twitter is a popular microblogging service where users create status messages called 'tweets'. People use 'tweets' to express and share their opinions and feelings about numerous topics. COVID-19 and its vaccines were no exception. This project is meant to identify sentiments of tweets over time and visualising them as aggregate data; to do that machine learning methods will be used.

1.2 **Literature Review**

Dubey aimed at analysing the sentiments of emoticons of the people during the COVID-19 pandemic. They collected country wise tweets from 11th of March to 31st of March 2020 using twitter's API and Rtweet package. They took data from 12 countries including Belgium, India, Australia, Netherland, Spain, United Kingdom, Italy, Germany, France, the USA, Switzerland, and China . After which they compared the number of tweets of positive and negative sentiments from each country. They found out that only China, the USA, and Switzerland had greater than 50% tweets of negative sentiment. They also analysed the use of emoticons in each country and found out that the USA, France, and China had the highest number of tweets with anger. Switzerland had the highest number of tweets with sadness and fear [1].

Wang et al. presented a system for real-time Twitter sentiment analysis of the ongoing U.S. presidential election. They used Twitter Firehose API which has no real limit to public Twitter post data compared to the Twitter API. They also used expert-curated rules and keywords to get a full and accurate picture of the online political landscape. Their work evaluates public sentiment changes in response to emerging political events and news as they unfold [2].

Go et al. put together different machine learning algorithms and techniques. Classifiers used include Naive Bayes which they built from scratch, Maximum Entropy, and Support Vector Machine. They explored the performance of different feature extractors unigram, bigram, negate as a feature, and part of speech features. They also explored what happens when a neutral sentiment is thrown in to the mix [3].

Naseem et al. collected tweets from the Twitter API using 'tweepy'; labelled them as positive, negative, or neutral using TextBlob. Then after pre-processing these tweets, they used TF-IDF Vectorizer to extract features and as classifiers they use Support Vector Machine, Naive Bayes, and Random Forest Classifier. They concluded that the population favoured the lockdown and stay at home order in February; however, their opinion shifted by mid-March [4].

Nemes and Kiss used a Recurrent Neural Network to classify tweets based on sentiments. They developed a model to search for connections between words and tagged them as positive or negative sentiment and compared the results to those of TextBlob's classification model [5].

Shamrat et al. conducted a sentiment analysis regarding COVID-19 vaccinations in 2021. After they processed their data they used K-Nearest Neighbour classifier algorithm to perform classification. They used TextBlob to get their learning polarity values for sentiments. They classified data into three classes: negative, neutral, and positive. From their results, it is seen that Pfizer vaccine shows 47.29% positive, 37.5%

negative, and 15.21% neutral, Moderna vaccine shows 46.16% positive, 40.71% negative, and 13.13% neutral, AstraZeneca vaccine shows 40.08% positive, 40.06% negative, and 13.86% neutral sentiment [6].

Hasan et al. compared the performances of various machine learning algorithms based on their performance on attacks and anomalies on the Internet of Things systems. The algorithms used are: Logistic Regression, Support Vector Machine, Decision Trees, Random Forest, and Neural Networks. Their performance was evaluated with various metrics, such as: accuracy, precision, recall, f1 score, and area under the Receiver Operating Characteristic Curve (AUC). They have got 99.4% test accuracy on Decision Trees, Random Forest, and Neural Networks. Though they have the same accuracy in other metrics Random Forest resulted in better values [7].

Minaee et al. have reviewed over 150 distinct deep learning models for text classification. They have also provided a summary of over 40 popular datasets created with text classification in mind. Finally, they have analysed the performance of different deep learning models on popular benchmarks [8].

Nowak et al. have demonstrated the superiority of LSTM RNNs over other algorithms for text classification on three datasets: Spambase Data Set, Farm Advertisement, and Amazon book reviews [9].

Sharfuddin et al. have reviewed the Recurrent Neural Network Bi-directional LSTM model and tested it on Bengali text. Achieving a 85.67% accuracy, while other traditional machine learning methods such as Support Vector Machine, Decision Tree Classifier, and Logistic Regression resulted in significantly lower accuracy rate ranging from 60% to 69% [10].

CHAPTER TWO

DATA

2.1 Overall Framework for Training

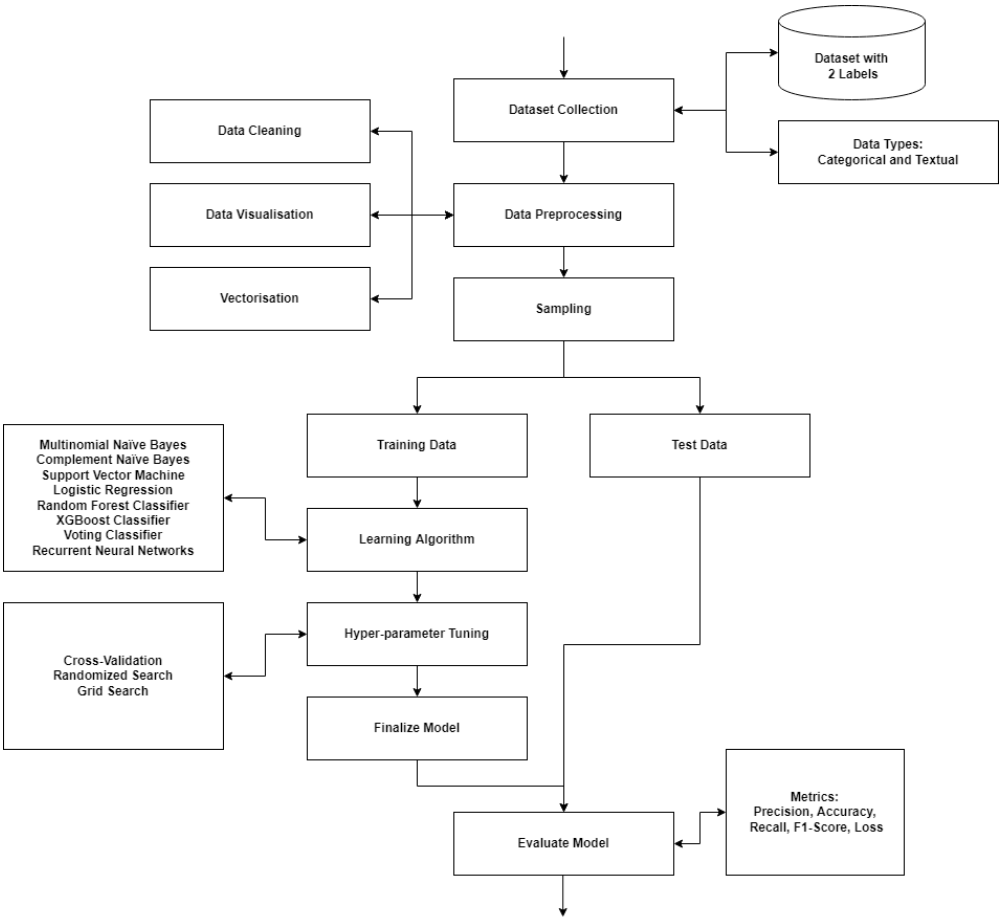


Fig. 1 Overall framework for training.

The chart above depicts the process of training and validating a model on a high level. The first process is the dataset collection process. In this process, one of two labelled datasets is collected for training.

The dataset has two types of data Categorical and Textual, which represent the labels and tweet data. In the next process, data pre-processing is applied to the dataset to clean the tweets of unnecessary information to make use of only the relevant data, visualise the data collected, and vectorize the data because computers can only work with numbers.

2.2 Dataset

The dataset that is used in this study is the Stanford's 'Sentiment140' dataset [11] which consists of 1.6 million tweets all labelled as either 0 or 4 representing negative and positive emotion respectively. There are equal numbers of negative and positive tweets in this dataset.

Table 1 Overview of Features

Feature	Example
target	0
id	1467810672
date	Mon Apr 06 22:19:49 PDT 2009
flag	NO_QUERY
user	scotthamilton
text	is upset that he can't update his Facebook by texting it…

2.3 Feature Selection

Feature selection is the process of selecting a subset of relevant features for use in model construction [12].

Feature selection techniques are used to simplify models to make them easier to interpret by others and shorten model training times. In the dataset there are columns that are irrelevant for training the model. These columns are id, date, flag, and user. Since it's unreasonable to expect to extract any insight about the emotion of tweets with these columns, they will be removed from our data frame.

2.4 Data Cleaning

Before extracting features from each document with the vectorisation operation, the data needs to be cleaned for training and analysis. In this study's case, urls, emojis, punctuations, mentions are unnecessary, so they need to be removed. Stop-words like 'a' or 'the' can also be removed before feeding into the vectorizer, but the vectorizer is already capable of ignoring the stop-words. Also, to cut off details from each word and only get the word bases, stemming operation is applied.

2.5 Visualisation of Data

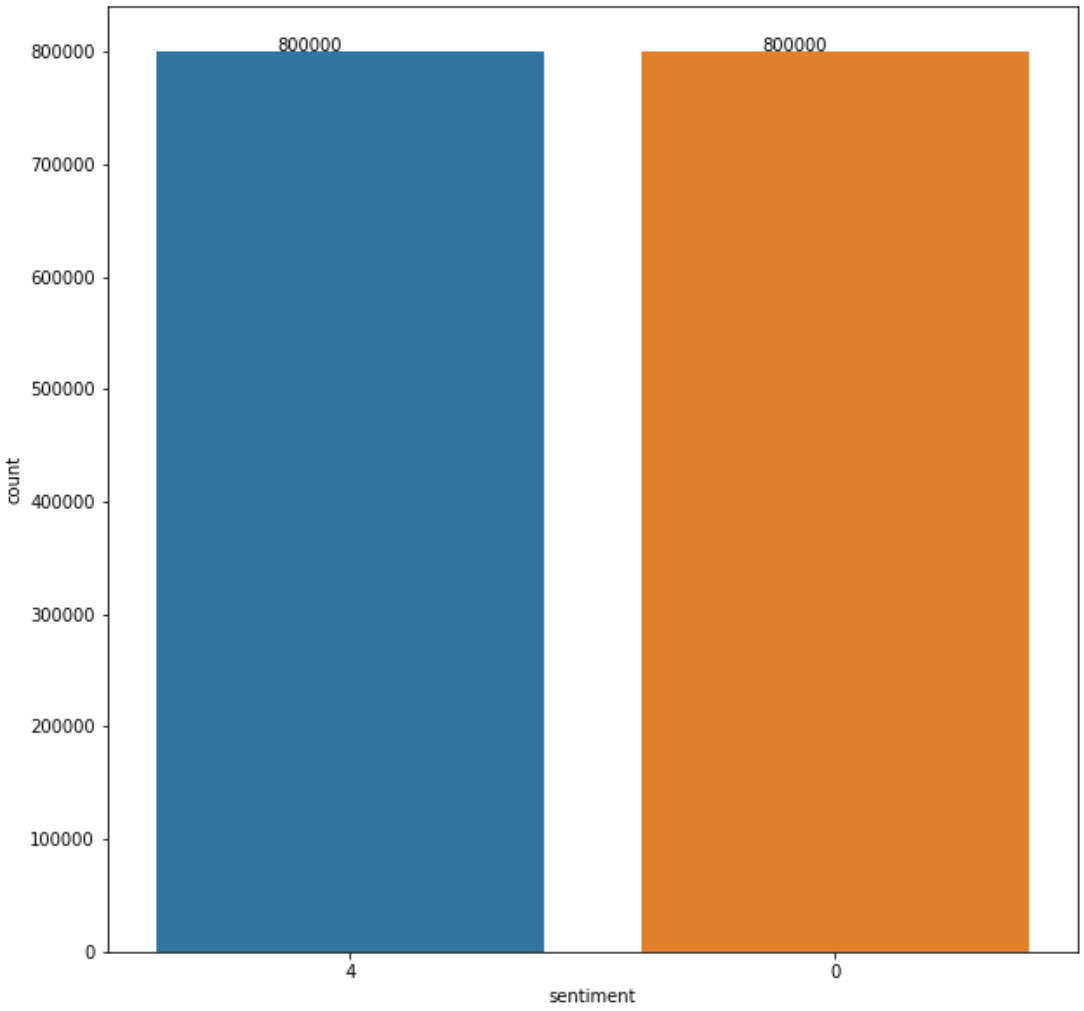


Fig. 2 Class count in training dataset.

Fig. 3 Positive words cloud with their frequencies emphasised by size.

Fig. 4 Negative words cloud with their frequencies emphasised by size.

2.6 Vectorisation

Since computers only know how to work with numerical data, the collected text data needs to be converted into numerical data and to do that vectorisation is used. The text data will be converted into a vector space which represents the word frequencies. Of course, since words like ‘the’, ‘an’, and ‘to’ also called stop-words appear a lot in text,

they need to be removed from the corpus. First thing a vectorizer will do is create a vocabulary that represents words as numbers. This vocabulary could be composed of different n-gram ranges. N-gram ranges represent how many words should be in a single vocabulary index. For instance, if the sentence "I live in Turkey", were to be represented in a unigram model it would be ['I', 'live', 'in', 'Turkey']. Moreover, in the case of bi-grams the result would be ['I live', 'live in', 'in Turkey'], which gives more context and relation between words.

Taking unigram model as the vectorisation method, the word 'brilliant' could be encoded with number 309. This means that in the vector space 309th index will represent the word 'brilliant'. Each document in the corpus which just means each tweet in the dataset will have its own vector space. Each vector space will have the same shape, which means some of the documents will either have too many or too few encoded words in their vector space. The size of the words can be set to be the number of unique words in the entire corpus or an arbitrary number that is neither too small nor too big. Since not every word might show up in other documents it makes sense to set a fixed number for vector size to only have space for the most common words in the vocabulary of the corpus, which would make the computation faster and interpretation simpler.

Another topic to consider is when a number representing a word appears too many times despite not being a stop word, in this case we can use inverse document frequency to reduce the impact that word will have on our vectors.

The formula for Term-Frequency is as follows:

$TF(t,d) = \sum_{x \in d} fr(x,t)$, where t is the term, d is the document, and x is the individual words in the document. The function `fr` simply returns 1 when the word x in document d matches the term t, otherwise returns 0.

The formula for Inverse Document Frequency is as follows:

$IDF(t) = 1 + \log_e \frac{1+|D|}{1+|\{d:t \in d\}|}$, where |D| is the total number of documents in the corpus, t is the term, and d is the individual documents. The denominator is simply the number of documents with term t in them. This is the scikit-learn's implementation to prevent zero divisions.

Multiplying Term-Frequency and Inverse Document Frequency gives us a vector space that has reduced the impact of most frequent words as well as increasing the impact of the rare words.

The resulting TF-IDF formula is as follows:

$$TFIDF(t) = TF(t,d) * IDF(t)$$

After TF-IDF vectors are calculated they are then fed into a normaliser. So that each feature vector will have unit Euclidean norm.

The formula for the for this is as follows:

$V_{norm} = \frac{V}{||V||^2} = \frac{V}{\sqrt{{V_1}^2+{V_2}^2+\cdots+{V_n}^2}}$, where V is a vector space of one dimension which means it is the feature vector space of one document. The normalised feature vector is calculated by multiplying the feature vector with the inverse of the square root of the sum of the squares of the vector's features.

To give a practical example, consider two documents' vectorisation process.

Document A: The car is driven on the road.

Document B: The truck is driven on the highway.

For this example, the stop-words to be removed from the corpus are: 'is', 'on', and 'the'.

Document A: car drive road.

Document B: truck drive highway.

Table 2 TF-IDF calculation example

Word	TF(A)	TF(B)	IDF	TF-IDF(A)	TF-IDF(B)
car	1/3	0	1.4	0.63	0
truck	0	1/3	1.4	0	0.63
drive	1/3	1/3	1	0.45	0.45
road	1/3	0	1.4	0.63	0
highway	0	1/3	1.4	0	0.63

Calculation for one row:

The word ‘car’ appears once in document A of three words, so TF (‘car’, A) = 1/3.

The word ‘car’ does not appear in document A at all, so TF (‘car’, B) = 0.

The number of documents is 2 but per scikit-learn’s implementation 1 is added to it, the number of documents the word ‘car’ has appeared in the corpus is 1 but once again per scikit-learn’s implementation 1 is added. Taking the natural logarithm of the result and adding 1 once again. IDF(‘car’) = 1.4.

TF-IDF (‘car’, A) = 0.33 * 1.4 = 0.63, but it’s not yet normalised.

TF-IDF (‘car’, B) = 0 * 1.4 = 0.

$V_{norm} = \frac{[0.63]}{\sqrt{0.63^2+0^2+0.45^2+0.63^2+0^2}} = 0.63$.

2.7 Sampling

To measure the model’s performance, a labelled testing dataset is also needed. Since the ‘Sentiment140’ dataset has more than enough data for training, some of it can be

reserved for evaluating the model. Ideally, while splitting the data for training and validation, class proportions need to be preserved. Scikit-learn has different methods of achieving this. One such methods is called `train_test_split` which does exactly what is needed.

```
from sklearn.model_selection import train_test_split

x = df['tweet']
y = df['sentiment']

x_train, y_train, x_test, y_test = train_test_split(x, y, stratify=y,
test_size=0.25, random_state=42)

print(y_test.value_counts())

# output
# 4 200000
# 0 200000
```

In the code snippet x is the tweet column of the dataset and y is the sentiment column which has the labels. The output displays 200,000 tweets labelled as 4 which means positive, 200,000 tweets labelled as 0 which means negative. So, the `x_train` and `y_train` values will be used to train the model and `x_test` and `y_test` will be used to evaluate the model.

CHAPTER THREE

MODEL

3.1 Naïve Bayes

Bayes' Theorem describes the probability of occurrence of an event related to any condition.

$P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$, where P(A|B) is the probability of A given B, P(B|A) is the probability of B given A, P(A) and P(B) are the probabilities of A and B respectively. In this case P(B) is the previous data, also called evidence. While P(A) is the prior assumption, which is a random value to be tuned over epochs. The naïve assumption is that the algorithm considers the features that it's using to make the predictions to be independent of each other. So, when a new set of features is given to predict its corresponding class, it considers each of the feature's probability of resulting in a specific class and taking the product of all the feature's results.

3.1.1 Multinomial Naïve Bayes

Multinomial naïve bayes is a specialised version of naïve bayes that handles text documents using word frequency as its underlying method of computing probability, although tf-idf vectors are also known to work well. The distribution is parametrised by vectors $\theta_y = (\theta_{y_1}, \dots, \theta_{y_n})$ for each class y, where n is the number of features and θ_i is the probability $P(x_i|y)$ of feature i appearing in a document belonging to class y [13].

3.1.2 Complement Naïve Bayes

Complement naïve bayes is an adaptation of the standard multinomial naïve bayes algorithm that is particularly suited for imbalanced datasets. Specifically, CNB uses

statistics from the complement of each class to compute the model's weights. The procedure for calculating the weights is as follows:

$\hat{\theta}_{ci} = \frac{\alpha_{,}+\Sigma_{j:y_j \neq c} d_{ij}}{\alpha+\Sigma_{j:y_i \neq c}\Sigma_k d_{kj}}$, $\omega_{ci} = \log_e \hat{\theta}_{ci}$, $\omega_{ci} = \frac{\omega_{ci}}{\Sigma_j|\omega_{cj}|}$, where the summations are over all the documents j not in class c, d_{ij} is either the count or the tf-idf value of term i in document j, α_i is a smoothing hyperparameter like that found in MNB, and $\alpha = \Sigma_i \alpha_i$. The second normalisation addresses the tendency for longer documents to dominate parameter estimates in MNB. The classification rule is:

$$\hat{c} = \underset{c}{argmin} \sum_i t_i \, \omega_{ci}$$

[14] Which means a document is assigned to the class that is the poorest complement match.

3.2 Support Vector Machines

In support vector classification, n-dimensional feature vectors are placed on an n-dimensional space and a threshold (also called a hyperplane) is calculated to separate m classes. When this threshold is placed as having equal distance to the closest vectors, it is called a soft margin and the vectors inside this margin are called support vectors. Soft margin allows outliers in data to be misclassified to perform better on new data, which means that the classifier will be less sensitive to the training data to account for the outliers which means it will have high bias and low variance. While the training accuracy may be lower its overall performance on new data will be significantly better. The concept of a model being highly sensitive to the training data is called overfitting which should be avoided.

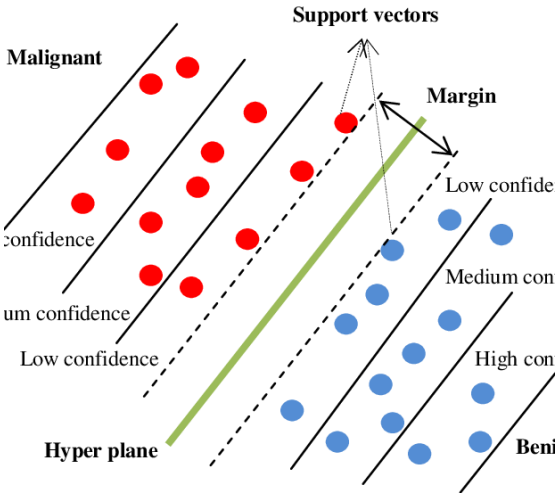


Fig. 5 SVM threshold representation

In the case where there are a lot of overlapping data points in the dataset, SVCs can't calculate a reasonable threshold to classify the data. This is called the linear separability problem. In which case support vector machines are used. SVMs are a set of supervised learning algorithms that can be used for classification. They increase the dimension of the data using a kernel function. For example, if the data is one-dimensional the polynomial kernel function with a degree of 2, will increase the dimension of the data to be 2-dimensional if the data was on point 0.75, its new point on the 2-dimensional space would be $(0.75, 0.75^2)$. On the new vector space, the SVC may be able to calculate a sufficient threshold for classification.

For text classification however, the linear kernel function may just work fine. The reason for that is that text data is often linearly separable. It is also advantageous because text data has a lot of features which would decrease the performance in other kernel functions.

3.3 Logistic Regression

In statistics, the logistic model is a statistical model that models the probability of one event (out of two alternatives) taking place by having the log-odds (the logarithm of the odds) for the event be a linear combination of one or more independent variables [15].

Logistic regression is a supervised learning algorithm used to predict a dependant categorical target variable, in this case the sentiment of a tweet. Logistic regression derives its name from the sigmoid function also known as the logistic function. The sigmoid function is an 'S'-shaped curve that stretches from 0 to 1 while never touching 0 or 1.

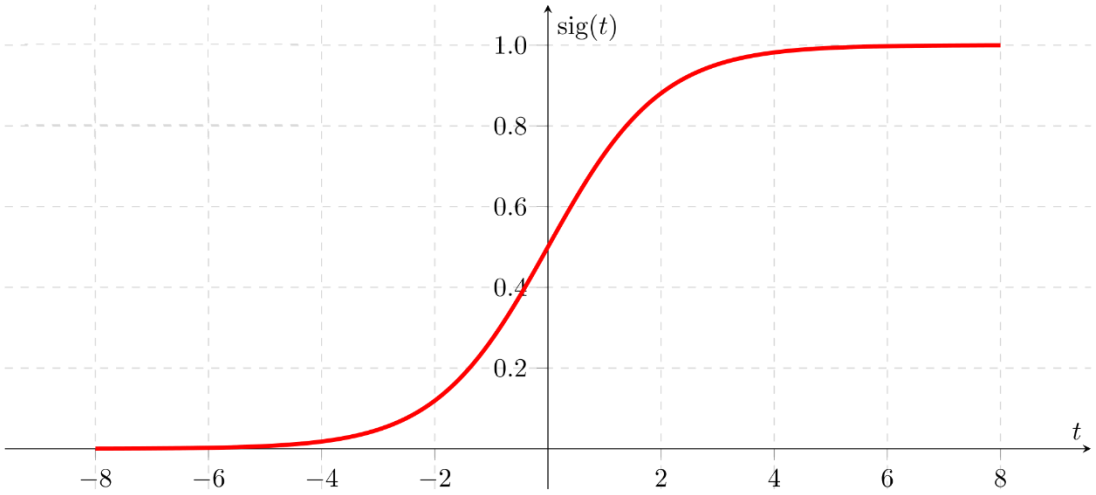


Fig. 6 The sigmoid function

$$sig(t) = \frac{1}{1 + e^{-t}}$$

There are different types of logistic regression but in a classification problem with two possible classes, binary logistic regression is used.

The differences between linear regression and logistic regression are that logistic regression predicts discrete values while linear regression predicts continuous values, instead of fitting a line over the data points like linear regression does, logistic regression fits the logistic function over the points which in this case the logistic function is the sigmoid function. Which results in extremely high confidence values where on one axis the value is very low or very high, whereas middling values may result in a near 50% chance of probability for both classes.

The classification is done by selecting a threshold for probability, if the probability is higher than the threshold the document is classified as positive, otherwise negative.

3.4 Decision Trees

Decision trees are popular methods for classification. They have a hierarchical tree structure which consists of a root node, branches, internal nodes, and leaf or terminal nodes. Root node is the node where the decision making process starts, branches are the options or the paths the decision may take, internal nodes are nodes that have children nodes and they represent another question, and finally the leaf or terminal nodes, as the name “terminal“ suggests, it is where the process ends, and a decision is made. The number of unique leaf nodes represent the number of different outcomes.

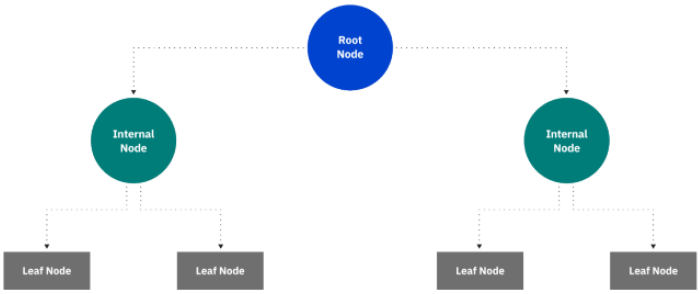


Fig. 7 Representation of a decision tree

Decision trees employ a greedy, top-down approach to the classification problem, meaning once a choice made for a particular node, it does not check for other possibilities which is good for classification [16].

3.4.1 Random Forest Classifier

Random forest is an ensemble of many other decision trees. Each decision tree is trained independently and the majority of the output of each of them is chosen for the classification. “Random“ part comes from the choosing of samples for each tree in the forest, all the samples are selected randomly. Each trees “votes” for a particular output and the majority is then picked by the ensemble. It works similar to the “Voting Classifier mentioned below. The important part is that the trees have no or low correlation between them so that their “vote” is unbiased.

3.4.2 Extreme Gradient Boosting (XGBoost) Classifier

Similar to Random Forest, Gradient Boosting Decision Trees like XGBoost are ensemble methods consisting of many other decision trees. The difference lies in how they are constructed and ensembled.

The term “gradient boosting” comes from the idea of “boosting” or improving a weak model by combining it with a number of other weak models in order to generate collectively stronger model. Gradient boosting is an extension of boosting where the process of additively generating weak models is formalized as a gradient descent algorithm over an objective function [17]. It sets targeted outcomes for the next model to minimize error. Targeted outcomes are based on the gradient of the error. It iteratively trains weaker trees, each iteration using the error values of the previous model to train the next.

XGBoost is a scalable and highly accurate implementation of gradient boosting that pushes the limits of computing power for boosted trees. In XGBoost, trees are built in parallel, unlike in regular gradient boosting decision trees where the trees are built sequentially.

3.5 Voting Classifier

The voting classifier model allows to combine several classifiers with different approaches to a problem by having them vote or use the average predicted probabilities on one sample at a time to predict the class labels. There are two voting methods: hard voting and soft voting.

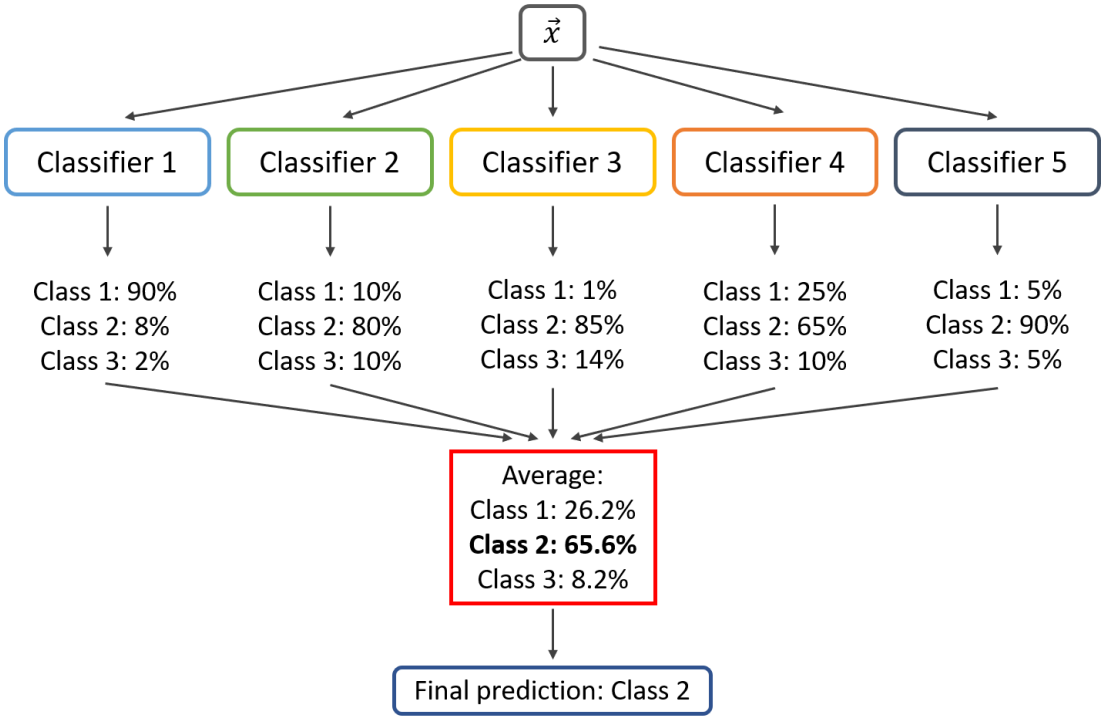


Fig. 8 Voting classifier

In hard voting (also called the majority voting), the classifiers make their own predictions, and the resulting label is the one that majority of the classifiers predicted. In the case of an even vote, voting classifier will make the prediction based on the ascending order of the classes.

In soft voting, the average of the weighted sum of probabilities for the classes is calculated, and naturally the highest one is the resulting prediction. That means some classifiers may have more say on what a sample should be classified as.

3.6 Recurrent Neural Networks

Recurrent neural networks (RNN) are a class of neural networks that is powerful for modelling sequence data such as time series or natural language [18]. They process sequences in this study's case tweets, one element at a time while keeping a memory (also called a state) of what the word before the current word was. People process

sentences as a whole and not individual words when composing a response, and recurrent neural networks aim to imitate that behaviour using gated recurrent units or long short-term memory layers.

The first step in using a recurrent network in our use case is to convert each individual word into vectors of n-dimensions, this is called word embedding, and in keras this can be achieved with the 'embedding' layer.

3.6.1 Embedding Layer

In the embedding layer each word is converted into a vector of n-dimensions, the values in the vector are called the weights, and generally they are initially random values to be tuned, but they can be pre-set using the values from another model already trained. If the weights are random, they will be updated with each epoch of training, this can be disabled if pre-trained weights are used, or not depending on the user's choice. The user may want to use pre-trained weights, but also fine tune them. In this study the weights will be starting off with random values.

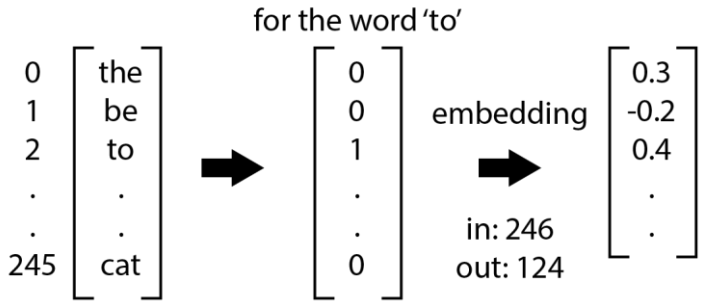


Fig. 9 Embedding layer visual

Embedding layer resembles a lookup table in that each index in the vector will have corresponding weights. Consider a vector with a shape of 1×125000, which means it is a word vector that has 125,000 random values. This vector will be multiplied with an embedding vector of shape $125000 \times n$, where n is the output dimension of the

embedding layer. The resulting vector will have a shape of $1 \times n$ which will be the output of the embedding layer for that word's corresponding node.

3.6.2 Long Short-Term Memory Layer

LSTMs are neural networks in and of themselves which differ from regular recurrent neural networks. They are designed to amend for situations where RNNs fail. The main idea is to keep the previous term's gradient (loss) in the memory for a long time.

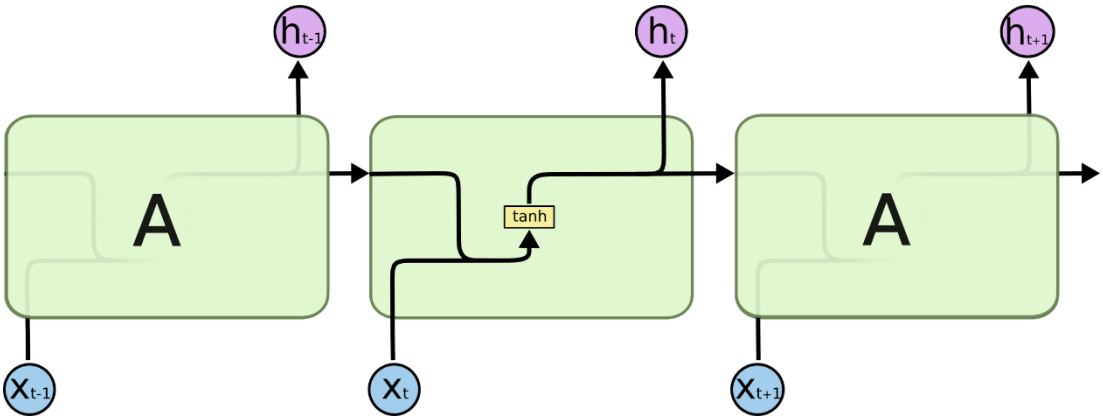


Fig. 10 A simple RNN

In simple RNNs the repeating modules have a rather simple structure like a single tanh layer. While LSTMs have this same repeating structure, what each module contains differ from RNNs [19].

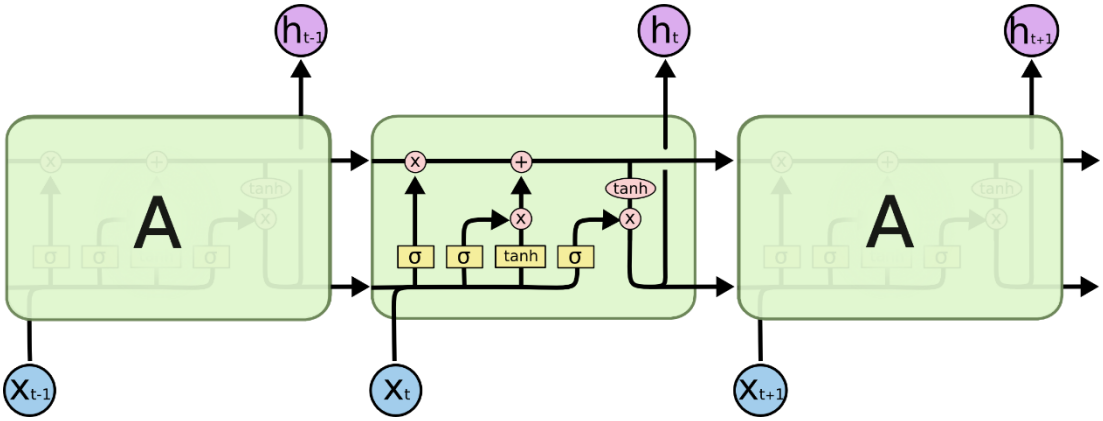
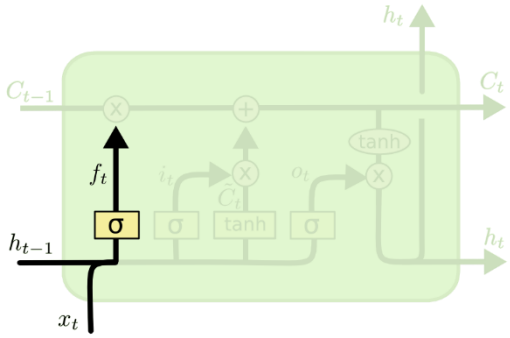


Fig. 11 An LSTM structure

LSTM modules have four layers inside. Each arrow-line transmits a vector, the pink nodes represent the pointwise operations for that time step, yellow boxes are activation

layers, there being three sigmoid and one tanh layer, lines converging represents concatenation of vectors, while lines separating denotes duplicating.

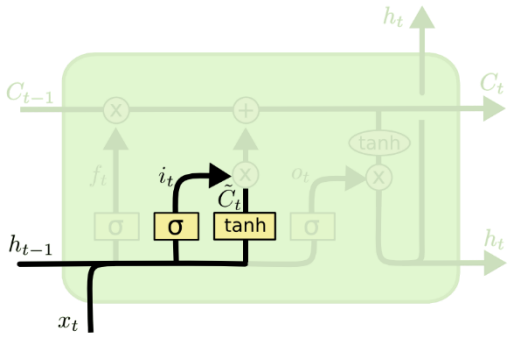


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Fig. 12 The forget gate layer

The first layer is called 'the forget gate layer'. It looks at the output from the previous time step h_{t-1}, and the input for the current time step x_t, and since it is a sigmoid layer, it outputs a value between 0 and 1 for each number in the previous cell state C_{t-1}. The number denotes how long this information should be kept. In the formula W_f is the weights vector and $b_f$is the bias of the current layer.

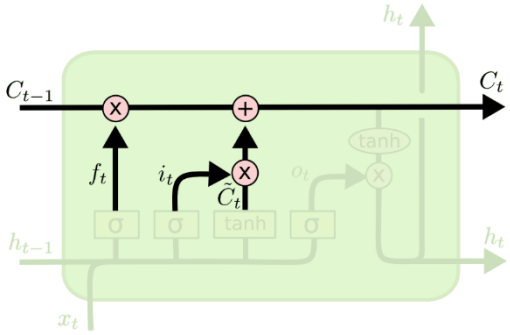
The next layer determines what new information is going to be stored in the cell state. This is done with two sub-layers, the first one is called 'the input gate layer', which is a sigmoid layer, and then the second one is a tanh layer that creates a new candidate vector $\tilde{C}_t$, that is a *candidate* that could be added to the next state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Fig. 13 The layers determining the candidate values to be stored

After which these two values will be combined to update the cell state.

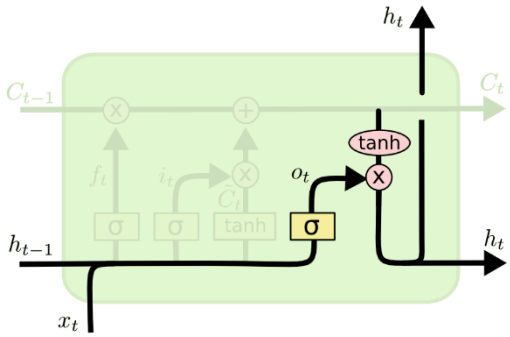


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Fig. 14 Update the cell state

Now that the network knows what information to forget and what not to forget, it can update the cell state, overriding the previous cell state C_{t-1}.The old state is multiplied with the output of the forget gate layer f_t, then the product of the input gate layer output and the candidate vector $i_t * \tilde{C}_t$, which means how important the new candidate vector values are.

In the last step, the output of the cell will be decided, it will be a concatenation of the previous output and the current input fed into a sigmoid layer, then multiplied with the new stored vector fed into a tanh layer.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Fig. 15 The processed output of the cell

An LSTM layer can be fed into a bi-directional layer to not only rely on the past data but on the future data as well.

3.6.3 Dense Layer

This layer is called dense because all the nodes in this layer are connected to all the nodes in the previous layer, which means it is *densely* connected. The values in the nodes of the dense layer are calculated as follows: $F(\sum_{i=0}^{n} w_i \times x_i + b_i)$, where F is the activation function, i denoting the previous layer's node, w meaning weights, x meaning the value of the node, and finally b is the bias of the last layer. So, the formula reads as the weighted sum of node outputs plus a bias for each node, fed into an activation function.

Activation functions transform the output of a neuron within a bound to decide whether the neuron should be activated or not. Some common activation functions are Relu, Sigmoid, and Tanh. Relu transforms the output to a value within 0 and ∞, which sets all the negative outputs to zero while keeping positive outputs the same. Sigmoid transforms the output to a value between 0 and 1. Lastly, Tanh transforms the output to a value between -1 and 1.

3.6.4 Gradient Descent

In neural networks the model is typically designed to have a low loss (also called cost) function value. The goal of gradient descent is just that. To achieve this the algorithm starts at a random point in the hyper-plane, the idea is to take repeated steps that iteratively decrease in size depending on the current loss value, to find the local minimum of the function.

The formula for the gradient descent is: $x_{n+1} = x_n - \gamma \nabla F(x_n), n \geq 0$, where γ is the learning rate of the loss function which scales the size of each step, F is the loss function, ∇F is the gradient of the function, and x is the loss value. x_0 is a random value that changes over time as the algorithm tries to find the minimum loss value on

the hyper-plane. The gradient is computed with an algorithm called ‘backpropagation’ [20].

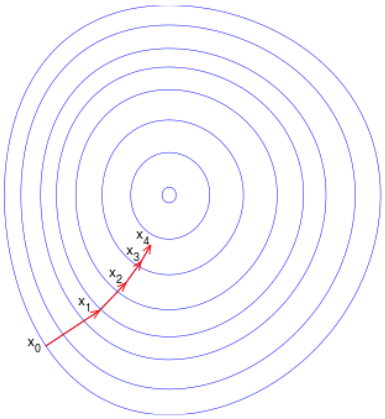


Fig. 16 Gradient descent

3.6.5 Backpropagation

Backpropagation is a widely used algorithm for training recurrent neural networks. In training a neural network, backpropagation computes the gradient of the loss function with respect to the weights of the network. It does so by starting from the last layer moving to the first layer [21].

The formula for backpropagation is as follows:
$g(x) = f^L(W^L f^{L-1}(W^{L-1} \dots f^1(W^1 x) \dots))$, where g(x) is the predicted output out the input x (vector of features), L is the number of the layer, f is the activation function. So, W^L means the weights of the layer L, while f^L means the output of the activation function of the layer L.

3.6.6 RNN Architecture

In the figure below, the representation for the recurrent neural network used for this study is shown. The layers are aligned vertically, and the number of nodes for each

layer minus the output layer are scaled down for simplicity. Each layer minus the output layer also shows the bias. The first layer in practice has 300 nodes. These are the outputs of the embedding layer which has an input dimension of approximately 100.000. The second and third layers are the LSTM layers which have 150 nodes each. Exactly half of the output of the first layer. The fourth layer is the dense layer with 32 nodes, and finally the binary output node.

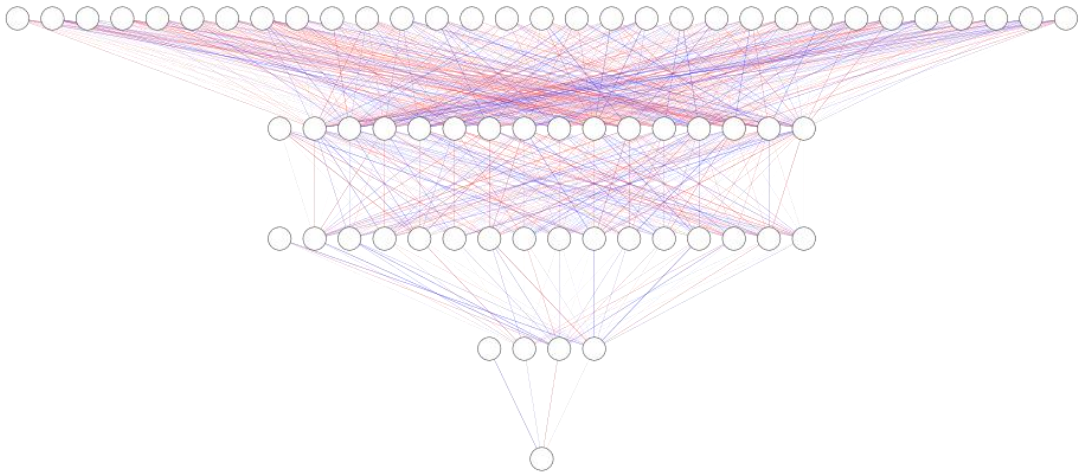


Fig. 17 RNN Architecture representation

CHAPTER FOUR

HYPER-PARAMETER TUNING AND CROSS-VALIDATION

4.1 Cross-Validation

While training a machine learning model, one problem that may arise is overfitting to the training data and becoming too specialised in the data given, such that the model performs worse on new unlabelled data. A way to combat this problem is called cross validation.

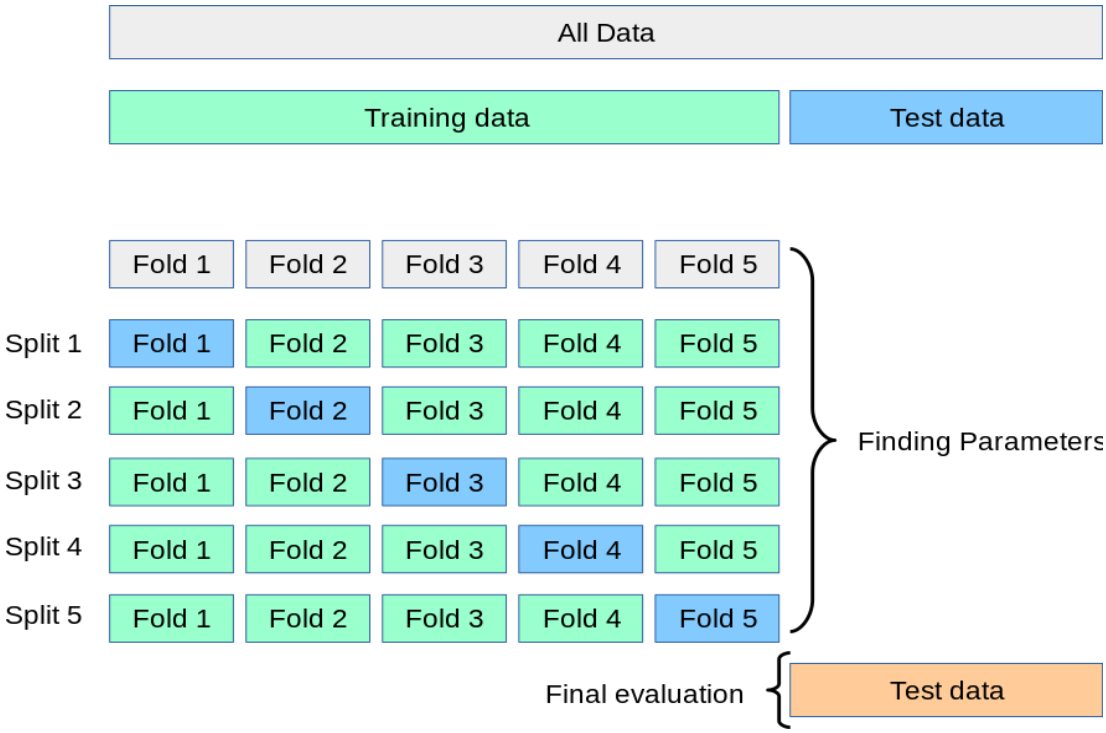


Fig. 18 Cross-validation

After splitting the dataset into training and testing data, the training data can be further split into k number of sets which are also called ‘folds’ and use only one fold for validation and re-train the model k times each time using a new fold as validation fold. When the training process finished the model then will be evaluated by the initial unseen test data. In this way, it was made sure that the model would never see the test data during the training process.

Hyper-parameters are parameters that are not directly learnt within the classifier itself. They are passed as arguments to vectorizers, transformers, models, and layers of models.

4.2 Randomised Search

One approach to try several combinations of these parameters is called randomised search. This approach samples a given number of candidates for parameters with a specified distribution (their possible values).

4.3 Grid Search

Another approach is called the grid search. Which exhaustively considers all the possible combinations of the given parameter space to find the best possible classifier.

CHAPTER FIVE

MODEL EVALUATION

5.1 Metrics

There are different metrics to measure the performance of a machine learning model on a given labelled dataset.

5.1.1 Confusion Matrix

Confusion matrix is a specific table layout that allows visualisation of the performance of an algorithm, typically a supervised learning algorithm [22].

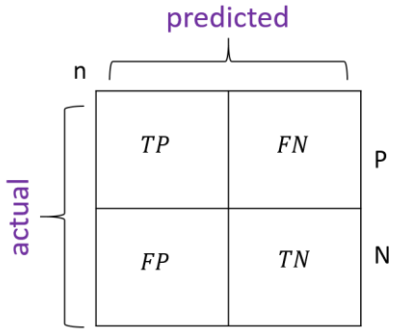


Fig. 19 The confusion matrix

5.1.2 Accuracy

The accuracy is defined as the correctly predicted documents divided by the total number of predictions.

$ACC = \frac{TP+TN}{TP+TN+FP+FN}$, where TP and TN are true positive and negative respectively. Just as well, FP and FN are false positive and negative.

5.1.3 Precision

Precision is defined as number of the correctly predicted documents belonging to one class divided by the total number of predictions of that class.

$PR = \frac{TP}{TP+FP}$, for the positive predicted documents.

5.1.4 Recall

Recall may be defined as the number of correctly predicted documents of one class divided by the total number of documents labelled as that class.

$RC = \frac{TP}{TP+FN}$, for the positive predicted documents.

5.1.5 Specificity

Specificity is a direct contrast to recall. In that it measures the number of correctly predicted negatives.

5.1.6 F1 Score

F1 score is the harmonic mean of precision and recall values.

$F1_p = \frac{2*PR_p*RC_p}{PR_p+RC_p}$, p meaning it's for positive values.

5.1.7 ROC and AUC

AUC means "Area Under Curve", the curve is the probability curve named "Receiver Operating Characteristic" or "ROC". It can be acquired by plotting the recall metric

against the specificity metric at assorted thresholds. The higher this value is the better is our model.

CHAPTER SIX

EXPERIMENTAL STUDY

6.1. Training Results

Using the data set containing 1.6 million labelled tweets. Cleaning the data, removing everything unnecessary. Splitting it into 1,200,000 rows of training and 400,000 rows of test data. Tokenising and vectorising the data with unigram and bi-gram models, and feeding the training data into various machine learning models gave the following results:

6.1.1 Unigram Model

Table 3 Results for the unigram model

Evaluation		**Classification Algorithms**						
Data	**Metrics**	**MNB**	**CNB**	**LSVC**	**LRG**	**RFC**	**XGB**	**VC-H**
Train	Accuracy	0.80	0.80	0.84	0.80	0.76	0.86	0.81
	Precision	0.80	0.80	0.84	0.80	0.77	0.86	0.81
	Recall	0.80	0.80	0.84	0.80	0.76	0.86	0.81
	F1-Score	0.80	0.80	0.84	0.80	0.76	0.86	0.81
Test	Accuracy	0.75	0.75	0.76	0.77	0.74	0.76	0.76
	Precision	0.755	0.755	0.765	0.775	0.75	0.77	0.765
	Recall	0.755	0.765	0.765	0.775	0.74	0.76	0.755
	F1-Score	0.755	0.755	0.755	0.775	0.74	0.76	0.76

In this column graph TR stands for the training accuracy of the models, the rest of the abbreviations are as follows:

- MNB: Multinomial Naïve Bayes
- CNB: Complement Naïve Bayes
- LSVC: Linear Support Vector Classifier
- LRG: Logistic Regression

- RFC: Random Forest Classifier
- XGB: Extreme Gradient Boosting Classifier
- VC-H: Hard Voting Classifier

It is seen that although the LSVC achieved the highest accuracy on the training data, LRG performed the best on the test data overall. Only giving up first place on precision for positive values.

6.1.2 Bi-gram Model

Table 4 Results for the bi-gram model

Evaluation		**Classification Algorithms**						
Data	**Metrics**	**MNB**	**CNB**	**LSVC**	**LRG**	**RFC**	**XGB**	**VC-H**
Train	Accuracy	0.82	0.82	0.88	0.83	0.76	0.86	0.83
	Precision	0.82	0.82	0.88	0.83	0.77	0.86	0.83
	Recall	0.82	0.82	0.88	0.83	0.76	0.86	0.83
	F1-Score	0.82	0.82	0.88	0.83	0.76	0.86	0.83
Test	Accuracy	0.77	0.77	0.77	0.79	0.74	0.76	0.78
	Precision	0.77	0.77	0.765	0.785	0.75	0.77	0.775
	Recall	0.77	0.77	0.765	0.785	0.74	0.765	0.775
	F1-Score	0.77	0.77	0.765	0.785	0.74	0.77	0.775

In the bi-gram model, a similar pattern to the unigram model can be seen. Just with overall higher values.

6.1.3 ROC-AUC

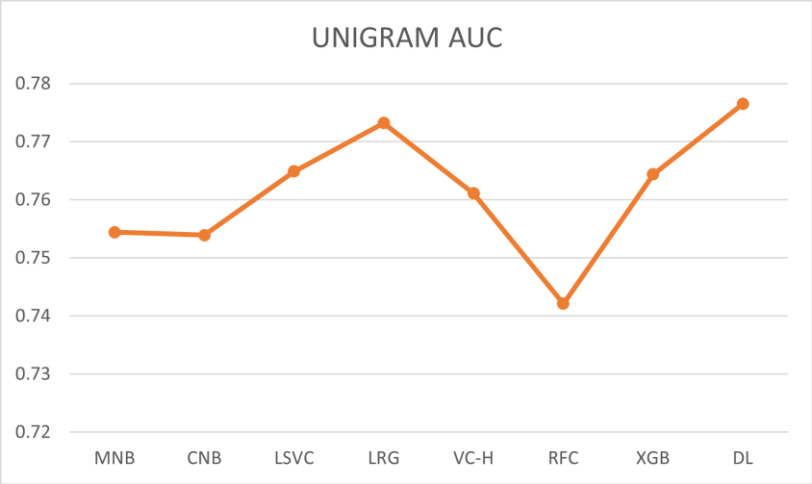


Fig. 20 ROC-AUC values for the UNIGRAM model

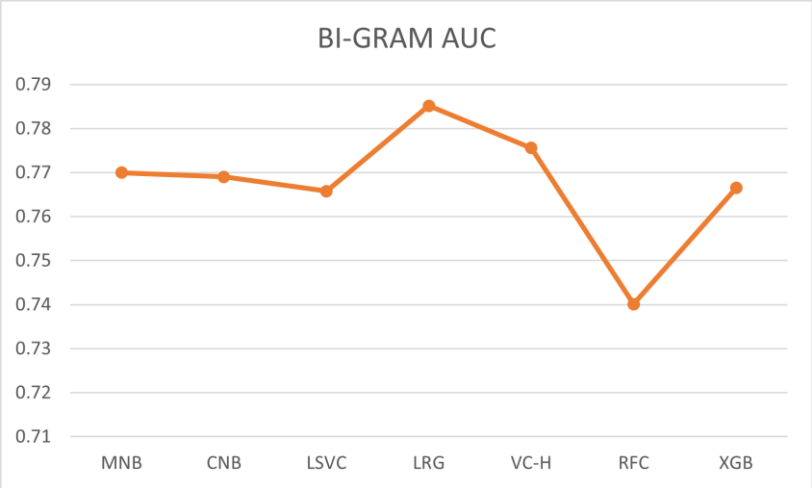


Fig. 21 ROC-AUC values for the BI-GRAM model

6.1.4 RNN Model

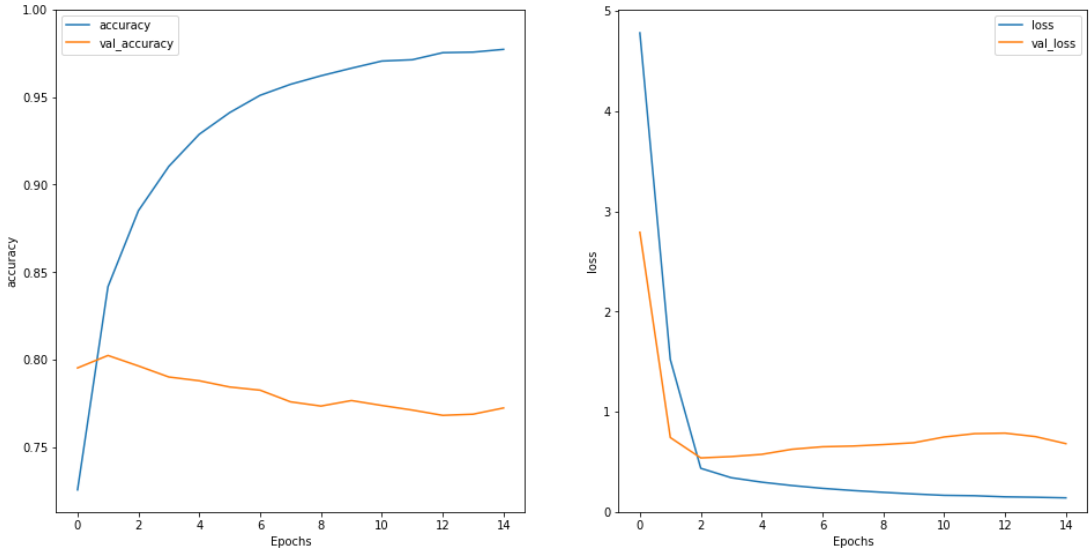


Fig. 22 Accuracy and loss values for the RNN model

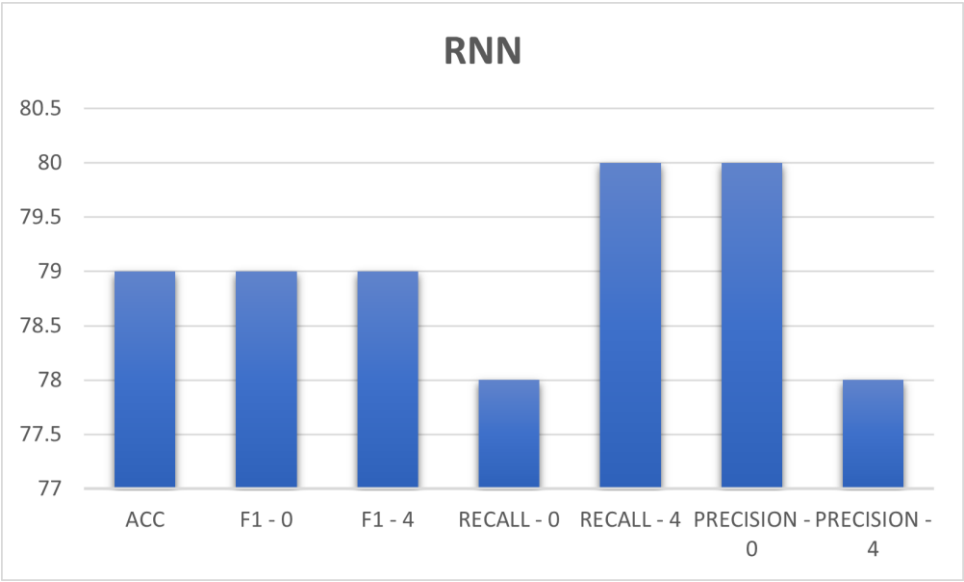


Fig. 23 Results for the RNN model

The objective of neural networks is to minimise the loss function. It seems the validation increased before slightly decreasing again after the seventh epoch. To prevent overfitting to the training data an early stopping call-back could be used.

6.2 Analysis on New Data

As shown in the graphs below, negative opinion about vaccines in general seems to be the majority from March 2020 to December 2021. Although that could also mean that people with negative opinions are more likely to express their opinion on social media. Whereas people with no concerns or fears are less likely to bother trying to encourage people to get vaccinated since that might seem like the obvious choice to protect oneself.

The number of tweets extracted by month:

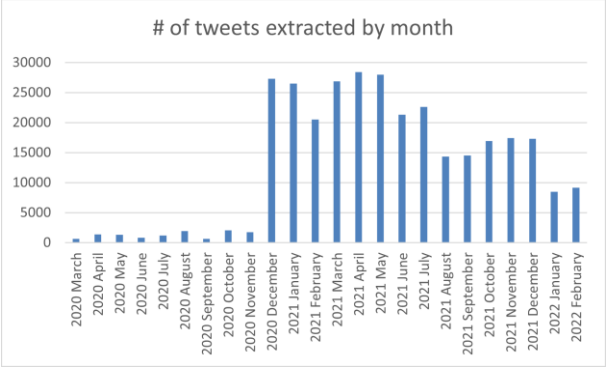


Fig. 24 Number of tweets analysed by month

The number of tweets extracted by vaccine:

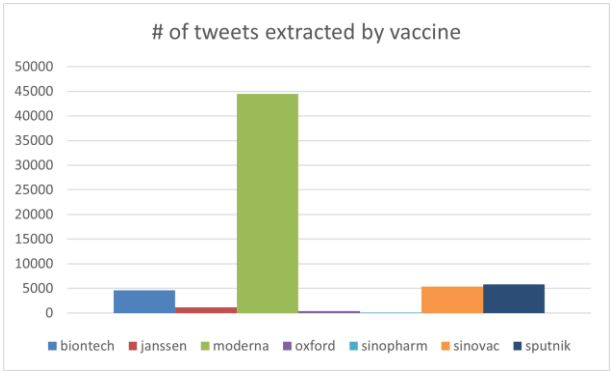


Fig. 25 Number of tweets analysed by vaccine

The results of the analysis per algorithm are as follows:

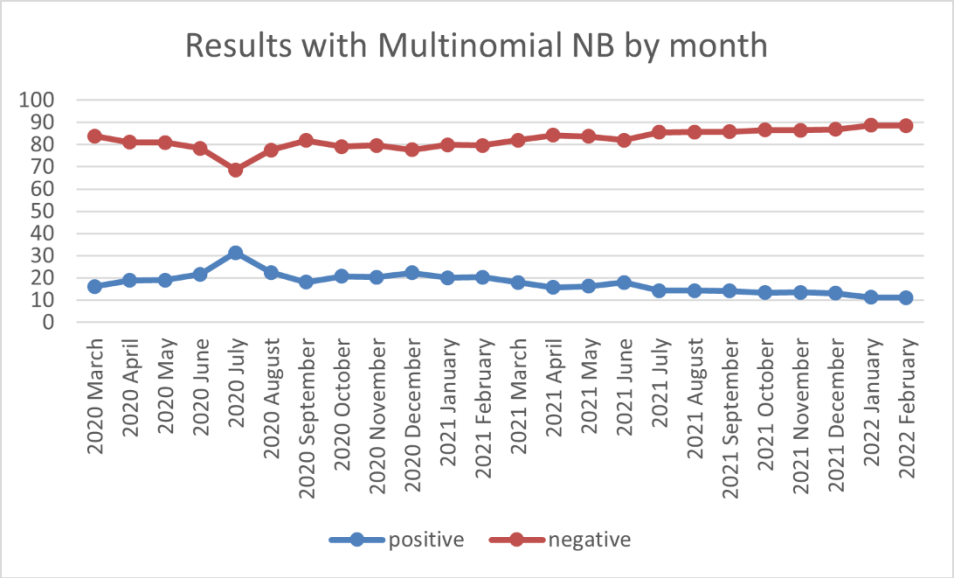


Fig. 26 Results with MNB by month

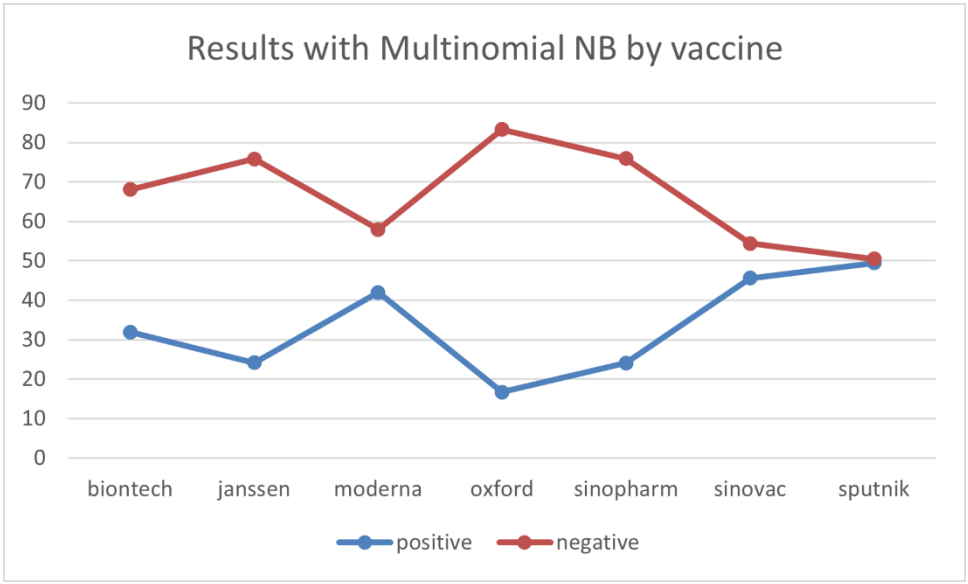


Fig. 27 Results with MNB by vaccine

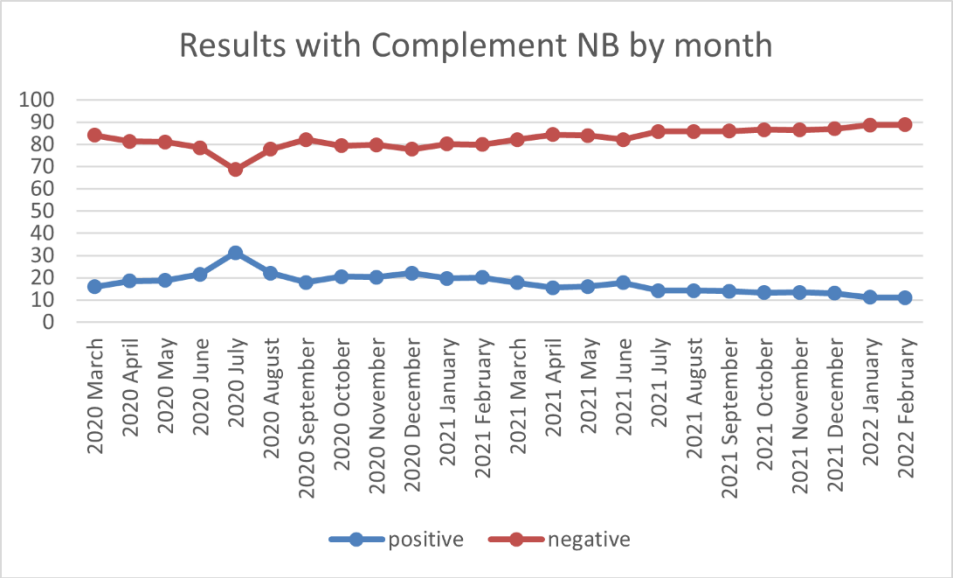


Fig. 28 Results with CNB by month

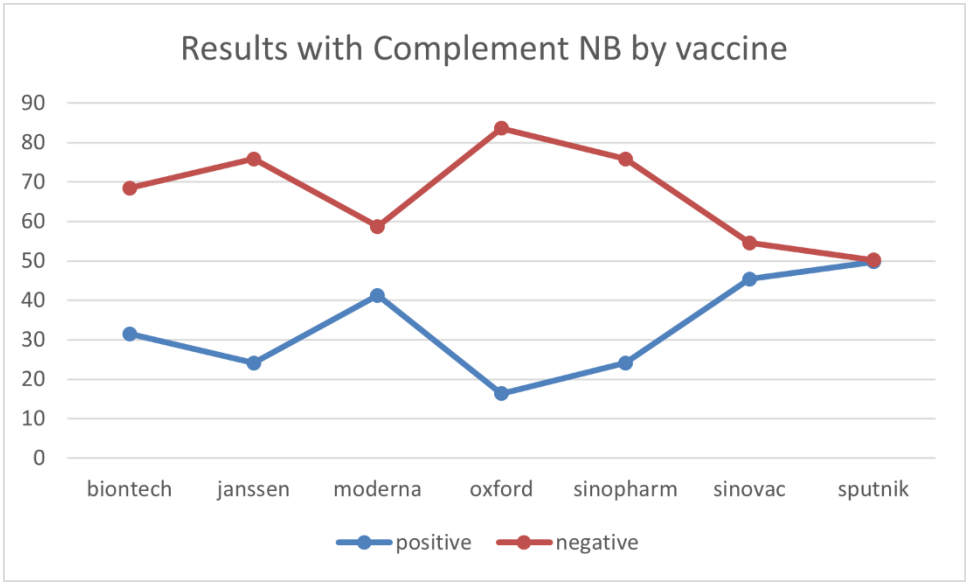


Fig. 29 Results with CNB by vaccine

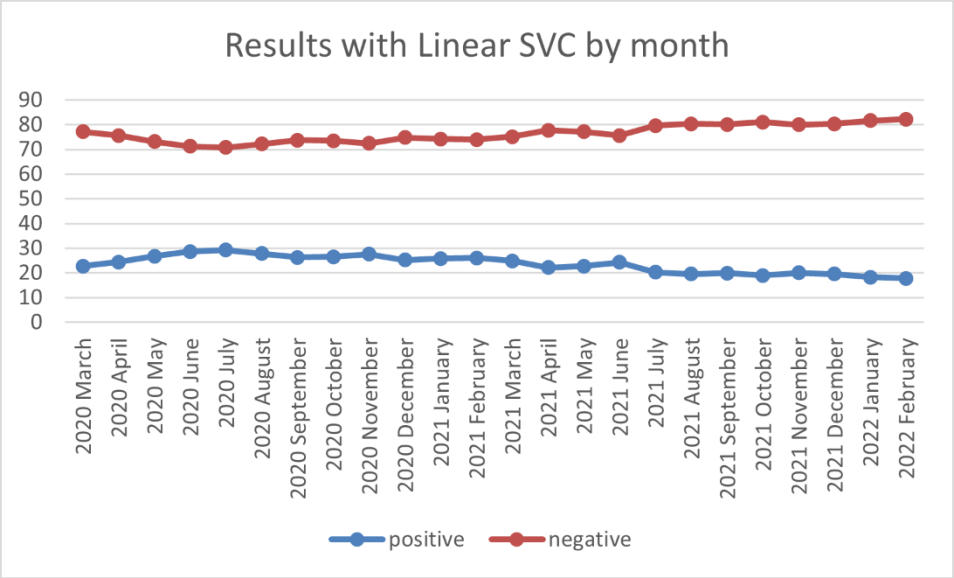


Fig. 30 Results with LSVC by month

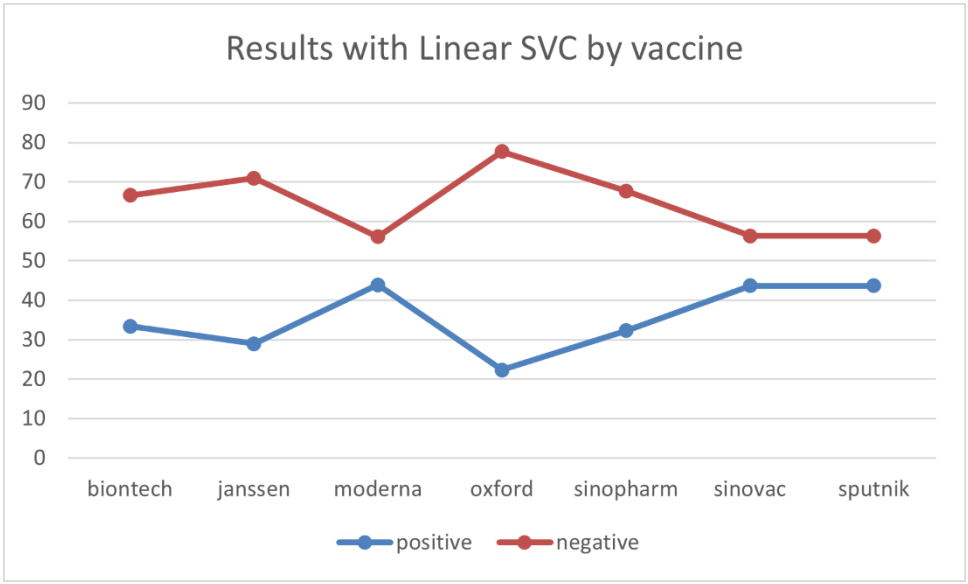


Fig. 31 Results with LSVC by vaccine

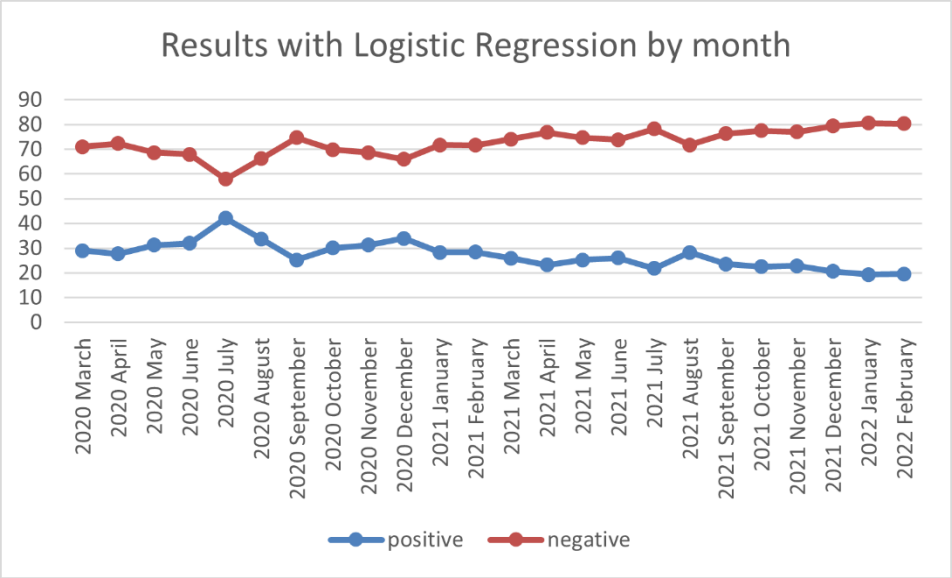


Fig. 32 Results with LRG by month

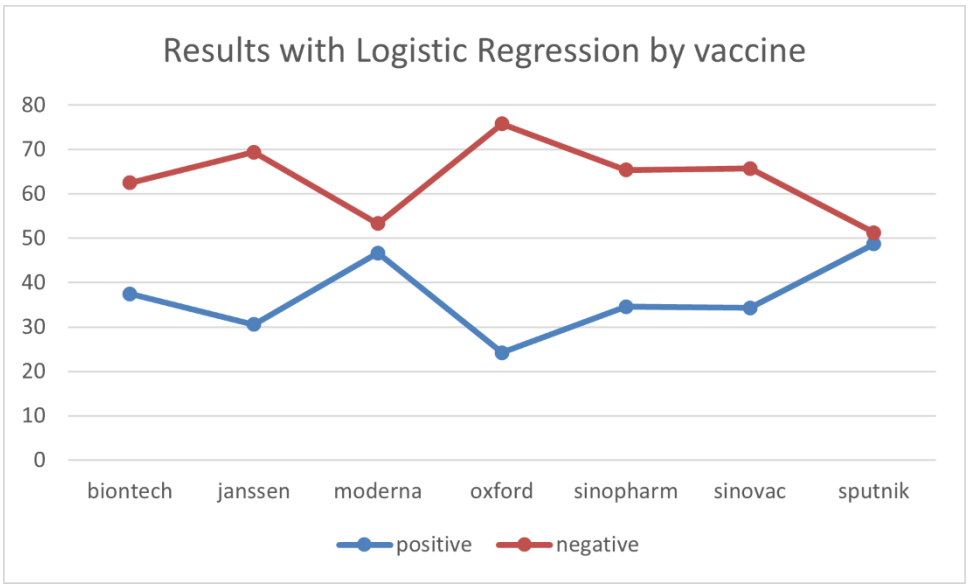


Fig. 33 Results with LRG by vaccine

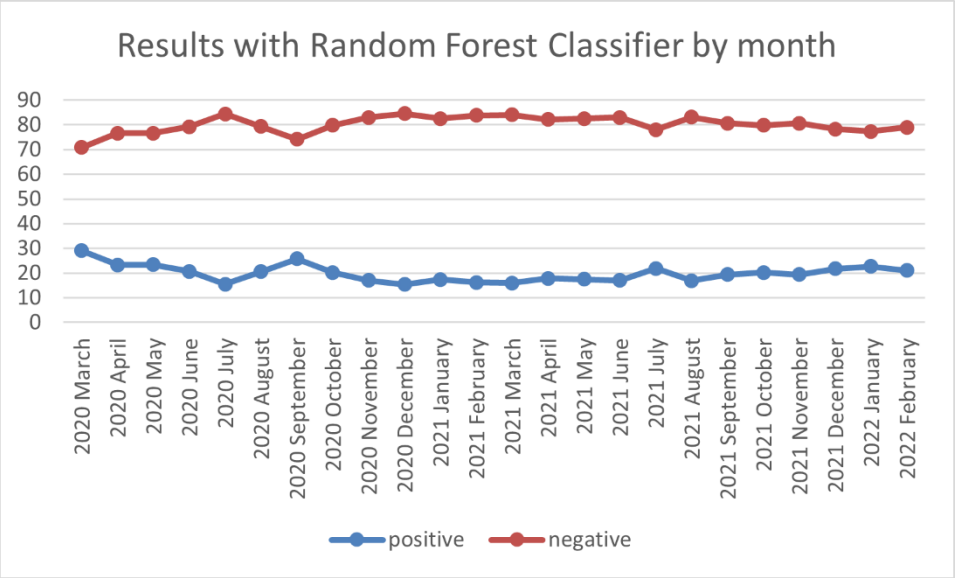


Fig. 34 Results with RFC by month

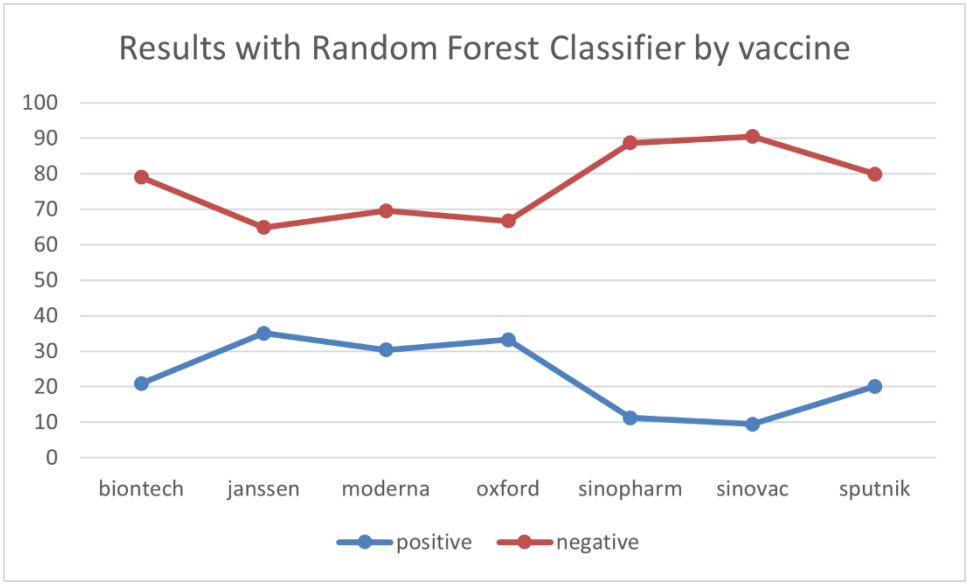


Fig. 35 Results with RFC by vaccine

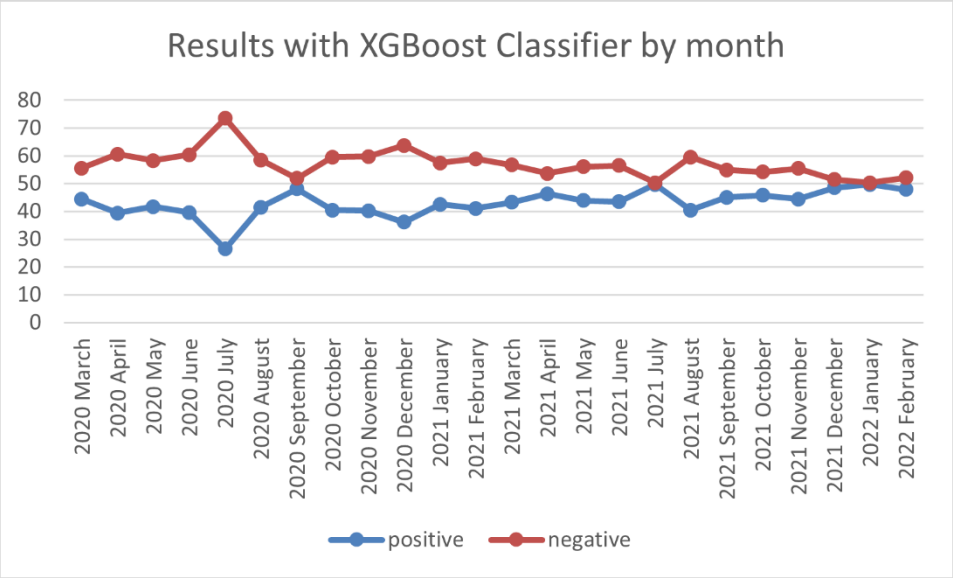


Fig. 36 Results with XGB by month

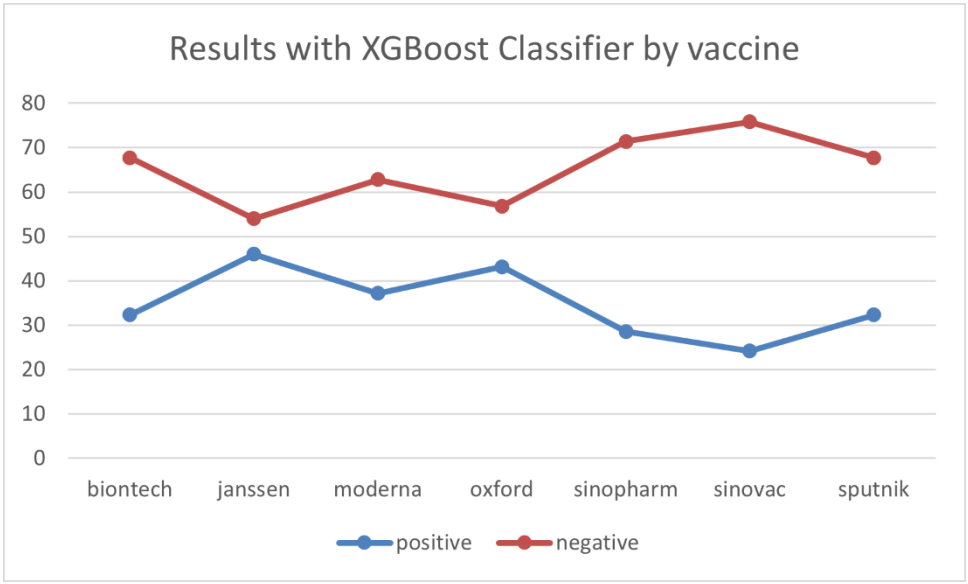


Fig. 37 Results with XGB by vaccine

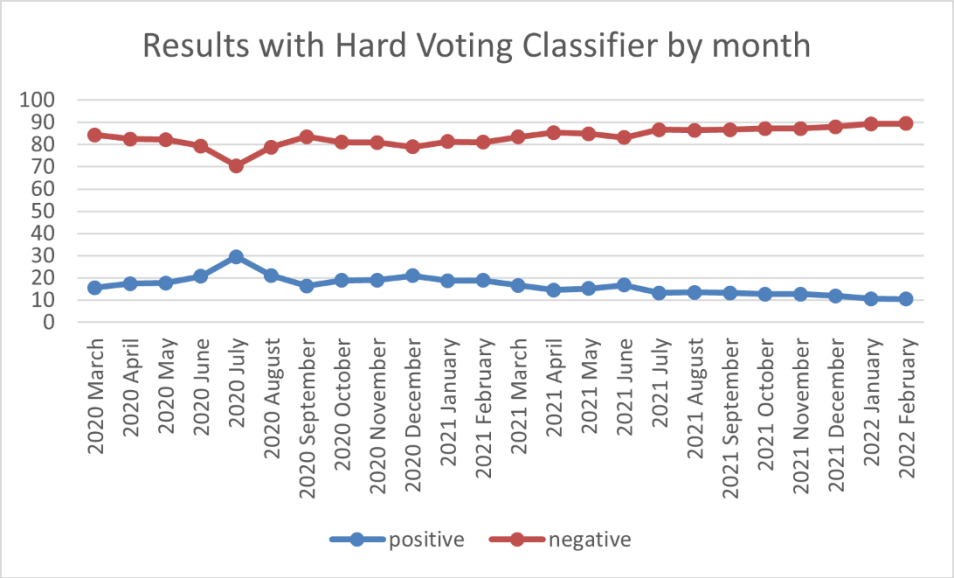


Fig. 38 Results with VC-H by month

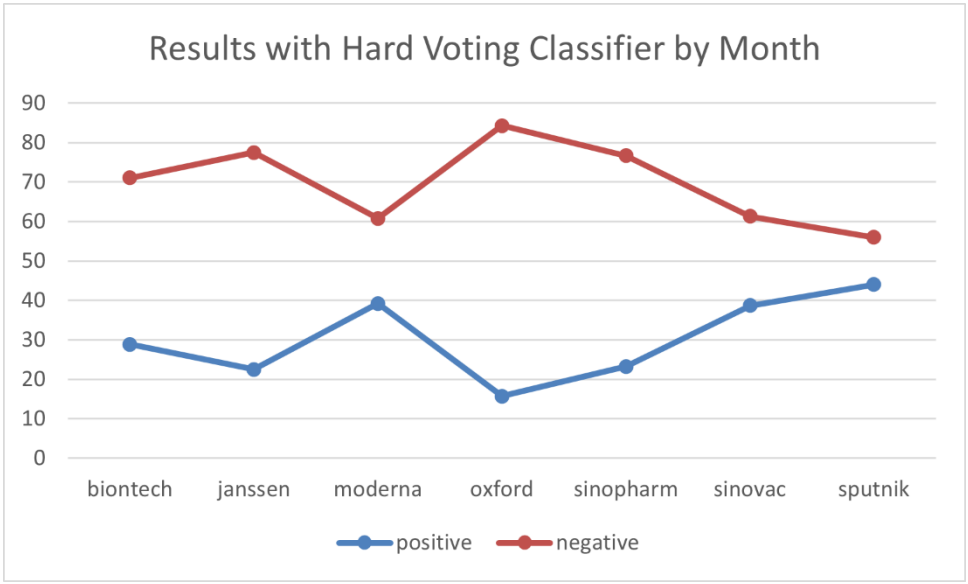


Fig. 39 Results with VC-H by vaccine

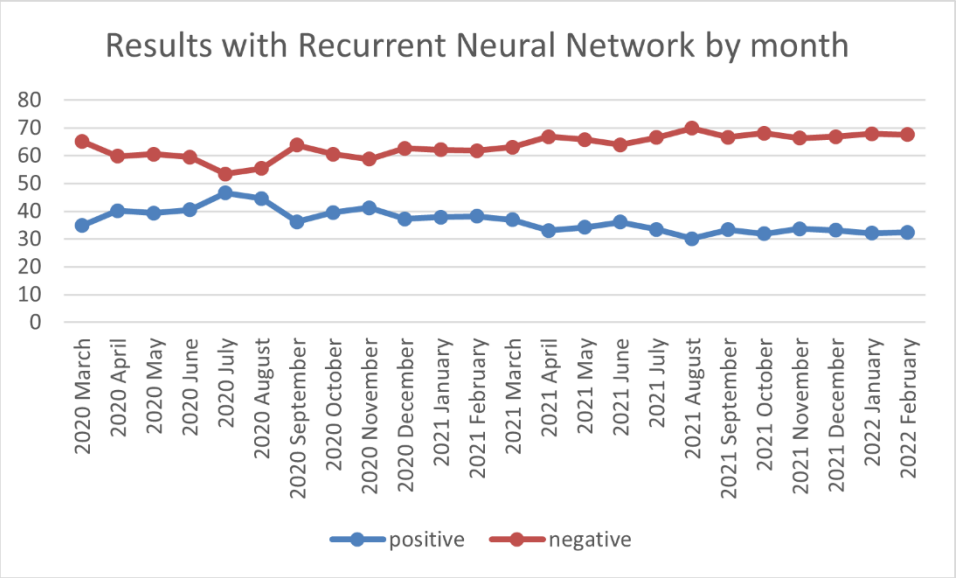


Fig. 40 Results with RNN by month

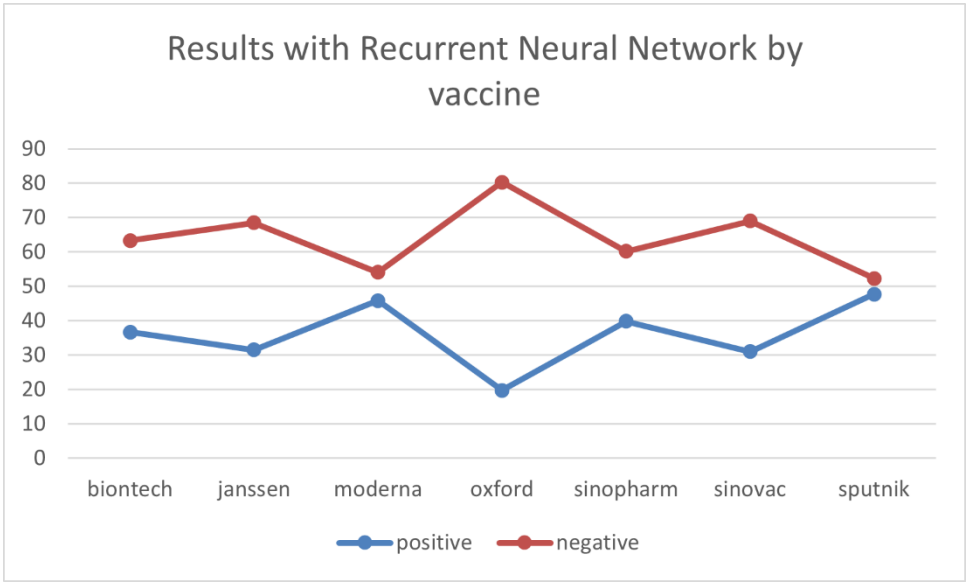


Fig. 41 Results with RNN by vaccine

6.3 Discussion

The difference between train and test metrics results on both unigram and bi-gram models, stem from the models being accustomed to the training data over their fitting period.

As it can be seen, the training results being higher for an algorithm doesn't mean that algorithm is going to do better on entirely new data. Linear Support Vector and Extreme Boosting Classifier are good examples of this. They had the highest training results, but both gave much lower results with the test data. This may mean that these algorithms are much more prone to overfitting than others, at least with the sentiment140 text dataset.

In the unigram model, the bi-directional LSTM Recurrent Neural Network achieved a whopping 98% accuracy on the training data. This is because it was allowed to train for more epochs than necessary, hence it grew more sensitive to the training data and the accuracy kept increasing while the test accuracy barely increased. Bi-directional LSTMs also consider the data around the current data point in both directions, meaning the words before and after a specific word will contribute to the classification together instead of taking all of them individually. The weights will be adjusted partially based on the previous and the next RNN's output.

For the unigram model, the best results for the metrics accuracy, f1-score, recall, and precision came from the deep learning method of bi-directional LSTM Recurrent Neural Network for both the training and the testing dataset.

For the bi-gram model, the RNN was out of the game and the best results overall came from Logistic Regression. As Logistic Regression relies on statistics, it works better the more data is fed into it, since the sentiment140 dataset had plenty of labelled data for Logistic Regression it performed well.

As for the analysis on new twitter data, it can be seen that negative opinion is the majority across all algorithms over the time period between March 2020 and 2022 February, also on the seven leading vaccines. Which is to be expected considering negative comments are more likely to be shared with others than positive opinion. It can also be noted that at the time interval twitter had much more anonymity than it is now, so people could share what they really thought without being reprimanded for it. It can be noted that similar algorithms returned similar results both by month and by vaccine, as it is clear in the results of Multinomial and Complement naïve bayes algorithms and decision tree classifiers in by vaccine results.

CHAPTER SEVEN

CONCLUSION

In this study, it was found that one would get more accurate results, if they utilized artificial neural networks, specifically recurrent neural networks with bi-directional long short-term memory architecture over traditional machine learning algorithms for text classification tasks. While the recurrent neural network performed better overall, it required more computational power to obtain results, alternatively logistic regression performed the best among the traditional machine learning methods as such it can be preferred over them when the data size is significant. It was also noted that on new data, extracted from twitter using ‘twint’, sentiment analysis results returned were expected to have majority to be negative sentiment and the analysis delivered on that account over 372,949 tweets.

REFERENCES

1. Dr. Akash D. Dubey, “Twitter Sentiment Analysis during COVID19 Outbreak” (2020).

2. Hao Wang, Doğan Can, Abe Kazemzadeh, François Bar, and Shrikanth Narayanan, “A System for Real-time Twitter Sentiment Analysis of 2012 U.S. Presidential Election Cycle” (2012).

3. Alec Go, Lei Huang, Richa Byahani, “Twitter Sentiment Analysis” (2009).

4. Usman Naseem, Imran Razzak, Matloob Khushi, Peter W. Eklund, Jinman Kim, “COVID Senti: A Large Benchmark Twitter Data Set for COVID-19 Sentiment Analysis” (2021).

5. Làszlò Nemes, Attila Kiss, “Social media sentiment analysis based on COVID-19” (2020).

6. F. M. Javed Mehedi Shamrat, Sovon Chakraborty, M. M. Imran, Jannatun Naeem Muna, Md. Masum Billah, Provita Das, and Md. Obaidur Rahman, “Sentiment analysis on twitter tweets about COVID-19 vaccines using NLP and supervised KNN classification algorithm” (2021).

7. Mahmudul Hasan, Md. Milon Islam, Md. Ishrak Islam Zarif, M.M.A. Hashem, “Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches” (2019).

8. Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao, “Deep Learning-based Text Classification: A Comprehensive Review” (2021).

9. Jakub Nowak, Ahmet Taspinar, and Rafal Scherer “LSTM Recurrent Neural Networks for Short Text and Sentiment Classification” (2017).

10. Abdullah Aziz Sharfuddin, Md. Nafis Tihami, Md. Saiful Islam, “A Deep Recurrent Neural Network with BiLSTM model for Sentiment Classification” (2018).
11. Sentiment140 Twitter Sentiment Analysis dataset, “https://www.kaggle.com/kazanova/sentiment140”.

12. Feature Selection, “https://en.wikipedia.org/Feature_Selection”.

13. Multinomial Naïve Bayes, “https://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes”.
14. Complement Naïve Bayes, “https://scikit-learn.org/stable/modules/naive_bayes.html#complement-naive-bayes”.

15. Logistic Regression,
“https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression”.

16. What is a Decision Tree? “https://www.ibm.com/topics/decision-trees”

17. What is XGBoost?
“https://www.nvidia.com/en-us/glossary/data-science/xgboost/”

18. Recurrent Neural Networks, “https://www.tensorflow.org/guide/keras/rnn”.

19. Understanding LSTM Networks
“http://colah.github.io/posts/2015-08-Understanding-LSTMs”.

20. Gradient Descent, “https://en.wikipedia.org/wiki/Gradient_descent”.

21. Backpropagation, “https://en.wikipedia.org/wiki/Backpropagation”.

22. Confusion Matrix, “https://en.wikipedia.org/wiki/Confusion_matrix”.